Copie par blocs d'octets avec dd

Sommaire

Introduction

La commande `dd` permet de copier tout ou partie d'un disque par blocs d'octets, indépendamment de la structuration du contenu du disque en fichiers et en répertoires.

Contrairement à la copie avec la commande cp ou la copie avec la commande tar, la copie avec dd permet de reproduire des zones de disque qui ne font pas partie d'un système de fichier : secteur de démarrage (le MBR), tables de partition, traces laissées sur le disque par des fichiers effacés etc. L'un de ses emplois les plus importants est donc **la création d'une copie de sauvegarde exacte de votre partition système et sa récupération**, par l'entremise d'un *live*-CD ou DVD en cas de pépin.

D'autres fonctions que la copie au sens strict, un peu « spéciales » mais parfois utiles, sont également rendues disponibles grâce à dd, comme vous le verrez dans les sections qui suivent : recherche dans les fichiers effacés, recherche dans la mémoire vive, création de disque virtuel etc.

ATTENTION LA COMMANDE dd PERMET DE FAIRE BEAUCOUP DE CHOSES INTERESSANTES, MAIS ELLE PEUT AUSSI ÊTRE DANGEREUSE, SOYEZ PRUDENT EN L'UTILISANT : UNE MANŒUVRE HÂTIVE POURRAIT RENDRE VOTRE SYSTÈME INUTILISABLE !!!

Vous trouverez un guide d'emploi de cette commande, raisonnablement clair, détaillé, bourré d'exemples - malheureusement en anglais... A la page :

Learn the dd command

Le contenu **d'une (petite !) partie** de ce lien a été traduit en français, ci-dessous, par **lebarhon** et révisé/remanié avec le concours de ptyxs.

Une autre page utile sur dd, hélas elle aussi en anglais, est :

Unix dd command

Pour d'autres techniques de copie, voir la page Copie de fichiers ou de répertoires.

Voir aussi La commande tar#Copie avec tar.

Pour la copie de sauvegarde ou de synchronisation, on pourra regarder, par exemple, ce que permet un logiciel comme Unison.

Les bases

La structure de la commande a la forme générale suivante :

```
dd if=<source> of=<cible> bs=<taille des blocs> skip= seek= conv=<conversion>
```

source représente les données à copier, *cible* est l'endroit où les copier.

`bs` est habituellement une puissance de 2, pas inférieure à 512, représentant un nombre d'octets (par exemple : 512, 1024, 2048, 4096, 8192, 16384, mais cela peut être tout nombre raisonnable).

Attention !! Si vous inversez la *source* et la *cible*, vous pouvez perdre beaucoup de données. Cette caractéristique a inspiré le surnom de `dd` : le Destructeur de Données !!

Le paramètre bs

Comment choisir `bs` (*block size* = taille des blocs) ? Un paramètre *bs=* correctement choisi accroitra significativement la vitesse d'exécution.

Sur du matériel moderne (moins de 5 ans) *bs=4096* est un bon pari.

On peut aussi avoir : *bs=16065b* qui peut être meilleur. *bs=32130b* encore meilleur. Ces deux dernières tailles de bloc correspondent à des nombres entiers de cylindres. Un cylindre en mode LBA = 255 têtes * 63 secteurs par piste = 16065 secteurs = 16065 * 512 octets = 16065b. Le **b** signifie ** 512* (512 octets étant la taille d'un secteur). 32130b représente un bloc de deux cylindres. Lorsque vous utilisez des tailles de blocs représentant des nombres entiers de cylindres, vous n'avez jamais à vous soucier de la copie de la dernière fraction de bloc car les partitions sont toujours faites d'un nombre entier de cylindres. Les partitions ne peuvent pas contenir de cylindres partiels. Un cylindre comprend 8 225 280 octets.

Histoires de têtes et de cylindres...

Pour bien comprendre les notions originelles de tête, cylindre et secteur, auxquelles l'auteur fait appel dans le paragraphe précédent, vous pourrez lire ces deux petites pages très éclairantes :

- http://fr.wikipedia.org/wiki/Disque_dur#G.C3.A9om.C3.A9trie

- http://www.redhat.com/docs/manuals/enterprise/RHEL-4-Manual/fr/admin-guide/s1-storage-data-addr.html

D'autre part, il faut bien comprendre que les notions de cylindre et de tête sous-jacentes au mode d'adressage LBA, sont « abstraites » et déconnectées de la réalisation matérielle du disque (votre disque n'a sûrement pas 255 têtes réelles...).

Pour mémoire :
Un cylindre en mode LBA = 255 têtes * 63 secteurs par piste (chaque tête lit une piste pour un cylindre donné) = 16 065 secteurs = 16 065 * 512 octets = 16065**b**. Le **b** signifie « * 512 ».
32130b représente la taille d'un bloc de deux cylindres.
Lorsqu'on utilise des blocs comptenant un nombre entier de cylindres, on n'a jamais besoin de se préoccuper du fait que la dernière partie d'un bloc ne soit pas copiée, car les partittions sont faites d'un nombre entier de cylindres. Les partitions ne peuvent pas contenir des cylindres partiels. Un cylindre compte 8 225 280 octets.

Les blocs et les octets peuvent être suivis par les suffixes multiplicateurs suivants (les valeurs numériques représentent des nombres d'octets) : **c** 1, **w** 2, **b** 512, **kB** 1000, **k** 1024, **MB** 1000*1000, **M** 1024*1024, **GB** 1000*1000*1000, G 1024*1024*1024

Ainsi :

```
dd if=/dev/sda of=/dev/sdb bs=1GB
```

utilisera des blocs de taille un gigaoctet.

bs=4b donnera à dd un bloc de taille de 4 secteurs de disque. 1 secteur=512 octets.

bs=4k indiquera à dd d'utiliser un bloc de taille 4 kiloctets. J'ai trouvé que *bs=4k* est le plus rapide pour la copie de disques sur une machine moderne.

Copier une partition de disque dur sur un autre disque dur

```
dd if=/dev/sda2 of=/dev/sdb2 bs=4096 conv=notrunc,noerror
```

sda2 et *sdb2* sont des partitions. Vous voulez copier *sda2* dans *sdb2*. Si *sdb2* n'existe pas, `dd` commencera au début du disque et le créera.

Faites attention à l'ordre de *if* et *of*. Vous pouvez écrire un disque vierge sur un disque utilisé si vous êtes désordonné.

Cloner un disque dur en entier

```
dd if=/dev/sda of=/dev/sdb conv=notrunc,noerror
```

Dans cet exemple, *sda* est la source. *sdb* est la cible. **N'intervertissez pas les sources et cibles ainsi désignées.** Étonnamment, beaucoup de personnes le font !
notrunc signifie « ne pas tronquer le fichier en sortie » (voir notrunc pour plus de détails).
noerror signifie de continuer en cas d'erreur. Normalement, dd s'arrête en cas d'erreur. Si vous vous interrogez sur un disque dur, si vous vous demandez s'il fonctionne ou pas, vous pouvez tenter de l'utiliser, à titre de test, comme lecteur source de la commande dd. Vous devriez obtenir une erreur s'il ne fonctionne pas. Les lecteurs cibles en revanche doivent être vraiment en très mauvais état pour provoquer une erreur due à dd.

Copier un grand disque sur un autre disque plus petit

La seule différence entre une grande partition et une petite partition, hormis la taille, est la table de partition. Si vous copiez *sda* vers *sdb*, un disque entier avec une seule partition, *sdb* étant plus petit que *sda*, alors vous devez faire :

```
dd if=/dev/sda skip=2 of=/dev/sdb seek=2 bs=4k conv=noerror
```

> Une autre technique pour traiter ce genre de situation, est proposée à la page 20 du fil en anglais. La technique proposée fait usage de commandes du type de `resize2fs`.

Les paramètres skip et seek

Skip (voir ligne de commande précédente) saute des blocs d'entrée au début du média (*sda*). *Seek* saute autant de blocs sur le média de sortie avant d'écrire (*sdb*). en agissant ainsi, vous laissez intacts les 4 premiers Ko sur chaque disque : vous ne voulez pas dire à un lecteur qu'il est plus gros qu'il n'est en réalité en écrivant une table de partition depuis un lecteur plus gros vers un lecteur plus petit.

dd d'une petite partition vers une plus grande : créer un fichier image de la partition

Les 63 premiers secteurs d'un lecteur sont vides, sauf le secteur 1, le MBR. Si vous copiez une partition plus petite vers une plus grande, la plus grande partition affichera sa taille correcte avec :

```
fdisk -l
```

Mais pas avec :

```
df -h
```

Cela parce que `fdisk` lit la table de partition et `df` lit l'info de format.

Si vous *dd* une plus petite partition vers une plus grande, la plus grande sera maintenant formatée identique à la plus petite et il n'y aura aucune place libre sur le lecteur. La façon de régler cela est de construire **un fichier image de la partition** :

```
dd if=plus_petite__partition of=/home/sam/plus_petite_partition.img
```

Montez l'image comme un lecteur en utilisant :

```
mount -o loop /home/sam/plus_petite_partition.img /mnt/directory
cd /mnt/directory
cp -r *
/mnt/plus_grande_partition_deja_partitionée_et_formatée_a_la_taille_desiree
```

La commande `cp` a un commutateur -r pour la copie récursive.

Alternative : Agrandir la partition après le dd

Une autre technique fait usage de la commande `resize2fs`. Cette commande va agrandir le système de fichiers à la taille de la partition :

1° Ne travailler que sur des partitions démontées

2° Ouvrir une console root (ou travailler depuis un live CD) et taper

```
e2fsck -f /dev/xx
```

où xx est évidemment le nom de la partition concernée, afin de vérifier l'intégrité du système de fichiers. Pour info, l'option -f force la vérification, sans se contenter d'un simple "check" du journal. De toutes façons, si vous ne ne faites pas, la commande suivante vous l'exigera avant de s'exécuter.

3° agrandir le système de fichiers :

```
resize2fs /dev/xx
```

dd d'une grande partition vers une plus petite

Maintenant, si vous copiez *sda3* vers *sda2* (plus petit que *sda3*), c'est différent. Ce que vous désirez faire est alors cela :

```
dd if=/dev/sda3 of=/dev/sda2 bs=4096 conv=noerror
```

La toute fin d'un lecteur contient généralement des zéros. Ainsi, si vous avez de la place dans *sda2* pour les données de *sda3*, les zéros de *sda3* sont tronqués (les blocs contenant uniquement des zéros sont remplacés par des chaines de cinq caractères astérisque) et c'est OK.

Dans ce cas, il ne faut pas utiliser *conv=notrunc*, qui copierait tous les blocs contenant uniquement des zéros tels quels, sans les tronquer.

Réaliser l'image ISO d'un CD

La commande :

```
dd if=/dev/hdc of=/home/sam/moncd.iso bs=2048 conv=notrunc
```

Les secteurs de CD ont une taille de 2 048 octets, vous faites donc ainsi une copie secteur par secteur. Le résultat sera un **fichier image du CD** sur le disque dur. Vous pouvez faire un `chmod a+rwx moncd.iso` pour permettre d'écrire dans l'image.

Vous pouvez monter l'image avec :

```
mkdir /mnt/moncd
```

écrire cette ligne dans le *fstab* :

```
/home/sam/moncd.iso   /mnt/moncd  iso9660 rw,user,noauto 0 0
```

Enregistrer le *fstab*, et réaliser le montage :

```
mount -o loop /mnt/moncd
```

Le système de fichiers est maintenant visible avec ses fichiers et dossiers dans le répertoire */mnt/moncd*. Vous pouvez éditer l'image autant que vous le souhaitez, le nouveau fichier sera */home/sam/moncd.iso*.

dd ne peut pas écrire dans un CD. Pour cela vous devez avoir recours à un logiciel de gravure ou à la commande `cdrdao`.

Effacer un lecteur

Si vous êtes inquiet au sujet d'espions avec des détecteurs supra-conducteurs à interférences quantiques, vous pouvez toujours ajouter une boucle *for* pour effacer le disque d'un niveau de sécurité gouvernemental : copier coller les deux lignes suivantes dans un éditeur de texte :

```
#!/bin/bash
for n in `seq 7`; do dd if=/dev/urandom of=/dev/sda bs=8b conv=notrunc; done
```

Maintenant vous avez un script shell qui exécute sept passes d'inscriptions aléatoires de caractères sur tout le disque.

Faites :

```
chmod a+x <fichierscript>
```

pour le rendre exécutable.

Créer une clé USB bootable

Pour créer une clé USB bootable: télécharger les 50 Mo de la distribution Debian ici

http://sourceforge.net/projects/insert/ (Lebarhon: On ne peut décemment pas écrire ça ici!!! o:))

Connectez la clé dans le port USB. Faites :

```
dmesg | tail
```

Regardez où est le nouveau lecteur, *sdb* ou quelque chose de similaire (sous une Mandriva il suffira de regarder, dans la fenêtre qui s'ouvre après insertion de la clé, le contenu de la rubrique *Emplacement*).

Faites :

```
dd if=/home/sam/insert.iso of=/dev/sdb ibs=4b obs=1b conv=notrunc,noerror
```

Maintenant modifiez le BIOS pour booter sur l'USB, connectez la clé et bootez la machine.

Copier seulement le MBR d'un disque dur

```
dd if=/dev/sda of=/home/sam/MBR.image bs=446 count=1
```

Ceci copiera les 446 premiers octets du disque dur dans un fichier.
ATTENTION : Si vous ne l'avez pas encore deviné, intervertir les arguments de *if=* et *of=*, dans la ligne de commande de dd intervertit le sens de l'écriture !

Sauvegarde de tout le disque

Pour ma part je travaille sur plusieurs machines, mais sur celle que j'utilise le plus j'ai deux disques durs SATA. Ils sont complètement identiques. Avant de faire quelque chose qui pourrait être dangereux, je démarre à partir de mon live CD helix et je lance :

```
dcfldd   if=/dev/sda    of=/dev/sdb/  bs=4096  conv=notrunc,noerror
```

(vous pouvez utiliser dd à la place de dcfldd) et je copie ainsi mon lecteur système de travail courant sur le lecteur *sdb*. Si je casse mon installation sur *sda*, je démarre avec le live CD et je fais :

```
dcfldd if=/dev/sdb  of=/dev/sda  bs =4096 conv=notrunc,noerror
```

Notez bien que *bs=4096* marche plus rapidement pour des machines ayant au moins 128Mo de RAM. dd utilise pas mal de tampons (*buffers*). Avec *bs=4096*, sur des machines modernes, la vitesse de transfert optimal pour les disques durs est atteinte.

Effacer toutes les données d'un disque dur

Vous devrez booter depuis un CD pour cela, http://www.efense.com/helix est un bon CD de boot (mais il y en a bien d'autres !). L'environnement de boot helix contient la version DoD de dd appelée `dcfldd`. Elle fonctionne de la même manière que dd, mais possède une barre de progression.

```
dcfldd if=/dev/zero of=/dev/sda conv=notrunc
```

Ceci est très utile pour rendre le lecteur presque comme neuf, cela permet de le débarrasser des virus, des chevaux de Troie etc... La plupart des lecteurs ont 0x0000ffh écrit en usine dans chaque secteur.

Ecrire par dessus toute la place libre d'une partition

C'est à dire rendre impossible la récupération des fichiers effacés.

```
dd if=/dev/urandom > fichieroccupanttoutlespacelibre
```

Quand dd dit : « plus d'espace libre sur le périphérique », tout l'espace disponible a été réécrit avec des caractères aléatoires. Ensuite, effacez le gros fichier avec :

```
rm
```

Quelques trucs pour les geek

Pour voir la mémoire vive

```
dd if=/proc/kcore | hexdump -C | less
```

utilisez PgUp, PgDn, flèche vers le haut, flèche vers le bas pour se déplacer dans les pages. `less` est mon éditeur préféré. Ou plutôt il le serait s'il permettait l'édition !

Quels systèmes de fichiers sont installés

```
dd if=/proc/filesystems | hexdump -C | less
```

Tous les modules chargés

```
dd if=/proc/kallsyms | hexdump -C | less
```

Table des interruptions

```
dd if=/proc/interrupts | hexdump -C | less
```

Depuis combien de temps fonctionne le système

```
dd if=/proc/uptime | hexdump -C | less
```

Partitions et tailles en Ko

```
dd if=/proc/partitions | hexdump -C | less
```

Etat de la mémoire

```
dd if=/proc/meminfo | hexdump -C | less
```

Créer un disque de sauvegarde

Nota Bene : *Dans les commandes qui suivent,* dcfldd *peut être remplacé par dd (*dcfldd *est un programme qui ajoute aux fonctionnalités de dd l'affichage d'une barre de progression).*

Je place deux disques identiques dans chacune de mes machines. Avant de faire toute chose pouvant être désastreuse, je fais :

```
dcfldd if=/dev/sda of=/dev/sdb bs=4096 conv=notrunc,noerror
```

et copie mon disque de travail en cours *sda* vers le disque *sdb*. Si je détruis l'installation sur *sda*, je n'ai qu'à booter avec le CD helix et :

```
dcfldd if=/dev/sdb of=/dev/sda bs=4096 conv=notrunc,noerror
```

et je retrouve chaque chose exactement comme avant qu'un truc idiot que j'essayais de faire ne fonctionne pas. Vous pouvez vraiment, vraiment apprendre Linux de cette manière, car vous ne pouvez absolument pas casser ce dont vous avez une copie exacte. Vous pourriez aussi envisager de faire une partition root séparée de `/home`, et rendre `/home` assez grand pour contenir la partition root, Vous pouvez alors faire :

```
dd if=/dev/sda2 (root) of=/home/sam/root.img bs=4096 conv=notrunc,noerror
```

pour réaliser un backup de root puis :

```
dd if=/home/sam/root.img of=/dev/sda2 (root) bs=4096 conv=notrunc,noerror
```

pour remettre l'image de root dans la partition root si vous avez semé la pagaille et ne pouvez plus lancer le serveur X, ou éditer `/etc/fstab`, et ne trouvez pas ce que vous avez fait de mal. Cela ne prend que quelques minutes pour restaurer une partition root de 15 Go depuis un fichier image.

Créer un fichier de 100 octets aléatoires

```
dd if=/dev/urandom of=/home/sam/mes_octets_aleatoires bs=1 count=100
```

Ici *urandom* est un générateur Linux de nombres aléatoires. *mes_octets_aleatoires* est un fichier. La taille de bloc est égale à 1 octet (*bs=1*) et on envoie 100 blocs, donc au total 100 octets (*count=100*).

gpg a besoin d'une *graine* (*seed*) aléatoire pour créer des clés. Créer un fichier de, disons, 4096 octets aléatoires, qu'on peut passer à *gpg*, permettra d'avoir une graine vraiment aléatoire.

`/dev/random` génère autant de bits aléatoires que la réserve entropique peut en contenir. Cela produit des valeurs hautement aléatoires pour des clés de cryptographie. Si davantage d'octets aléatoires sont nécessaires, le processus s'arrête jusqu'à ce que la réserve entropique soit remplie de nouveau (bouger la souris aide). `/dev/urandom` n'a pas cette restriction. Si l'utilisateur exige plus de bits que de présents à ce moment dans la réserve entropique, ils sont produits en utilisant un générateur de nombres pseudo-aléatoires.

Ecrire des données aléatoires par dessus un fichier avant de l'effacer

d'abord, faire :

```
ls -l
```

pour trouver la taille du fichier :

```
ls -l unfichier
-rw------- ... 3769 Nov 2 13:41 <filename>
```

dans ce cas elle est de 3769.

```
dd if=/dev/urandom of=unfichier bs=3769 count=1 conv=notrunc
```

Ceci écrira des caractères aléatoires par dessus la totalité du fichier .

Ainsi même quelqu'un qui inspectera le contenu du disque avec dd ne retrouvera pas son contenu.

Copier une partition de disque dans un fichier placé sur une partition différente

Attention!! Ne pas copier une partition sur la même partition.

```
dd if=/dev/sdb2 of=/home/sam/partition.image bs=4096 conv=notrunc,noerror
```

Cela créera un fichier qui est l'exacte réplique de la partition *sdb2*. Vous pouvez remplacer par *hdb*, *sda*, *hda* ou toute autre désignation de votre disque. Ou :

```
dd if=/dev/sdb2 ibs=4096 | gzip > partition.image.gz conv=noerror
```

Crée une archive *gzip* de la partition complète. Pour la restauration, utiliser :

```
| gunzip >
```

Pour *bzip2* (plus lent, plus petit), remplacer par *bzip2* et *bunzip2*, et nommer le fichier

```
.bz2
```

Restaurer une partition de disque depuis un fichier image

```
dd if=/home/sam/partition.image of=/dev/sdb2 bs=4096 conv=notrunc,noerror
```

De cette façon, vous pouvez avoir un grand disque dur et le partitionner, ainsi vous pouvez sauvegarder votre partition root. Si vous semez la pagaille dans votre partition root, vous n'avez qu'à booter depuis le CD helix (ou un live CD quelconque) et restaurer l'image.

Convertir un fichier tout en caractères majuscules

```
dd if=filename of=filename conv=ucase
```

Créer un lecteur virtuel

Le noyau Linux crée généralement nombre de disques virtuels que vous pouvez transformer en lecteurs virtuels. Vous devez d'abord remplir le lecteur avec des zéros comme ceci :

```
dd if=/dev/zero of=/dev/ram7 bs=1k count=16384
```

ce qui crée un disque virtuel de 16 Mo plein de zéros...

Ensuite :

```
mkfs.ext3 -m0 /dev/ram7 4096
```

place un système de fichiers sur le disque virtuel le transformant en lecteur virtuel.

```
debian:/home/sam # hdparm -t /dev/ram7
/dev/ram7: Timing buffered disk reads: 16 MB in 0.02 seconds = 913.92 MB/sec
(Débit de lecture du disque bufférisé: 16Mo en 0,02 secondes= 913,92 Mo/s)
```

Vous pouvez êtes tenté de ne mesurer le débit qu'une fois. Mais, en fait, il vaut mieux lancer plusieurs fois `hdparm -t /dev/ram7`, car `hdparm` est un peu difficile avec les lecteurs virtuels.

Vous pouvez monter le disque virtuel avec :

```
mkdir /mnt/mem
mount   /dev/ram7 /mnt/mem
```

Maintenant, vous pouvez utiliser le lecteur virtuel comme un lecteur matériel. Ceci est particulièrement remarquable pour travailler sur des gros documents ou en programmation. Vous pouvez copier le gros document ou le projet de programmation sur le lecteur virtuel, lequel sur ma machine est au moins 27 fois aussi rapide que *`/dev/sda`*, et chaque fois que vous enregistrez l'énorme document, ou que vous avez besoin de compiler, c'est comme si votre machine carburait au nitrométhane. La seule chose est que **le lecteur virtuel est volatile. Si vous perdez l'alimentation, ou éteignez, les données sur le lecteur virtuel sont perdues.** Utilisez une machine fiable, par temps clair, si vous utilisez un lecteur virtuel.

Copier la mémoire RAM dans un fichier

```
dd if=/dev/mem of=/home/sam/mem.bin bs=1024
```

Le périphérique :

```
/dev/mem
```

est votre mémoire système. **Vous pouvez en fait copier tout périphérique de type bloc ou caractère dans un fichier avec dd.**

Copies diverses

Vous pouvez en fait copier tout périphérique de type bloc ou caractère dans un fichier avec dd.

La capture de la mémoire sur un système rapide, avec bs=1024 prend environ 60 secondes, un disque dur de 120 Go environ une heure, un CD vers un disque dur environ 10 minutes, une disquette vers un disque dur environ 2 minutes.

Avec dd, les images sur vos disquettes ne changeront pas du tout. Si vous avez une disquette DOS bootable, et que vous la sauvegardez sur votre disque dur sous forme de fichier image, quand vous restaurez cette image sur une autre disquette, elle sera bootable.

dd est un excellent outil pour créer une image d'un CD d'installation de MS Windows. Lorsque vous faites une copie d'un tel CD, il y a un secteur qui possède une taille non standard. C'est le dernier secteur. dd ne remplit pas ce secteur, créant ainsi une copie indistinguable de l'original. Si vous gravez le CD, avec `cdrao`, le disque résultant sera une copie absolument exacte de l'original.

dd écrira sur la console si vous oubliez la partie of=/dev/output

```
dd if=/home/sam/monfichier
```

écrira le fichier monfichier dans la console.

Chercher dans la mémoire système

```
dd if=/dev/mem | hexdump -C | grep 'chaine-de-caractères-comprise-dans-le-fichier-non-sauvegardé-avant-la-perte-de-l'alimentation'
```

Effacer la RAM

Si vous avez besoin d'effacer vos traces rapidement, placez la commande suivante dans un script pour écrire des zéros par dessus la mémoire système. N'essayez pas cela pour le plaisir.

```
mkdir /mnt/mem
mount -t ramfs /dev/mem /mnt/mem
dd if=/dev/zero > /mnt/mem/bigfile.file
```

Cela écrira des zéros par dessus toute la mémoire non protégée, et gèlera la machine, vous aurez donc à rebooter (attention, cela empêche aussi le fonctionnement du journal du système de fichiers et pourrait corrompre le système de fichiers).

Lire des secteurs du disque

Si vous êtes curieux de savoir ce qu'il peut bien y avoir sur votre disque...

Lire le MBR

Si vous voulez voir à quoi ressemble un MBR, faites :

```
dd if=/dev/sda count=1 | hexdump -C
```

vous montrera le secteur 1, ou MBR. Le code de l'amorce et la table des partitions sont dans le MBR.

Lire la fin du disque

Pour voir la fin du disque vous devez connaitre le nombre total de secteurs dans le disque, et le disque doit être configuré avec le Maximum Adressable Sector (le secteur adressable maximum) identique au Maximum Native Address (= MNA, l'adresse native maximale). Le CD helix possède un utilitaire pour paramétrer cela correctement. Dans la commande dd votre valeur pour skip sera un de moins que la MNA du disque. Pour un disque Seagate SATA de 120 Go :

```
dd if=/dev/sda of=/home/sam/monfichier skip=234441646 default bs=512
```

Ainsi cela lit secteur par secteur, et écrit le dernier secteur dans monfichier. **Même avec l'adressage LBA, les disques sont lus « secrètement » en secteurs, cylindres et têtes.**Pour un cylindre donné il y a 63 secteurs par tête, et on compte 255 têtes par cylindre. Chaque disque possède un nombre total bien déterminé de cylindres. Le nombre total d'octets pas cylindre s'obtient donc par la multiplication suivante : 512x63x255= nombre d'octets par cylindre. 63x255= 16 065=nombre de secteurs par cylindre. 512 est la taille en octets d'un secteur. Avec 234 441 647 secteurs au total, et 16 065 secteurs par cylindre, vous obtenez 14 593.317584812 cylindres, un nombre qui n'est pas entier, et il y a alors quelques secteurs excédentaires qui ne constituent pas un cylindre entier. Ceci vous laisse avec 5 102 secteurs qui ne peuvent pas être partitionnés car une partition ne comprend que des cylindres entiers. C'est comme avoir une partie de personne, cela ne constitue pas vraiment une personne. Ainsi, qu'arrive t-il à ces secteurs ? Ils deviennent des **secteurs en surplus** après la dernière partition. Vous ne pouvez pas en principe les lire avec un système d'exploitation. Mais dd peut. C'est vraiment une bonne idée de vérifier ce qui peut être écrit dans les secteurs en surplus. Pour notre disque dur Seagate de 120 Go vous soustrayez du nombre total de secteurs (234 441 647) ceux en surplus (5 102) = 234 436 545 secteurs partitionables.

Pour y voir plus clair sur ces histoires de têtes, cylindres et secteurs, jetez un coup d'oeil plus haut.

Ceci écrit les 5 102 derniers secteurs dans monfichier. :

```
dd if=/dev/sda of=/home/sam/monfichier skip=234436545
```

Lancez *Midnight Commander* (`mc`) pour voir le fichier. Si il y a quelque chose dedans, vous n'en avez pas besoin pour quoi que ce soit. Dans ce cas vous devriez écrire par dessus des caractères aléatoires.
dd if=/dev/urandom of=/dev/sda bs=512 seek=234436545

Ecrasera les 5 102 secteurs en surplus sur notre notre disque dur Seagate de 120 Go

Vérifier une partie quelconque du disque

La commande :

```
dd if=/dev/sda of=/home/sam/monfichier bs=4096 skip=2000 count=1000
```

écrira dans *monfichier*, les 8 000 secteurs qui suivent les 16 000 premiers secteurs du lecteur.

Ben oui, quoi !! On saute (*skip*) 4096 * 2000 = 8 192 000 octets, d'accord ? Bon. Comme il y a 512 octets dans un secteur, on a sauté : 8 192 000/512 = 16 000 secteurs !!
Et ensuite on a envoyé (*count=1000*) 1000 blocs de 4096 octets, soit 4 096 000 octets, ce qui fait finalement 4 096 000/512= 8 000 secteurs, vous voyez qu'on s'y retrouve !!!

Vous pouvez ouvrir ce fichier avec un éditeur hexadécimal, en éditer une partie, et recopier cette partie sur le disque :

```
dd if=/home/sam/monfichier of=/dev/sda bs=4096 seek=2000 count=1000
```

Ainsi vous obtenez un **éditeur de disque**. Ce n'est pas le meilleur, mais il fonctionne.

Disquettes

Copier une disquette sur le disque dur

```
dd if=/dev/fd0 of=/home/sam/floppy.image bs=2x80x18b conv=notrunc
```

ou

```
dd if=/dev/fd0 of=/home/sam/floppy.image conv=notrunc
```

18b représente la taille en octets de 18 secteurs de 512 octets chacun, 80x multiplie par le nombre de cylindres, 2x multiplie par le nombre de têtes - au total 1474560 octets. Cela paramètre une unique requête de lecture de 1474560 octets sur /dev/fd0 et une unique requête d'écriture sur /home/sam/floppy.image.

Il faut comprendre que, dans le cas d'une disquette, pour chacun des 80 cylindres supposés, 2 têtes (fictives) lisent chacune une piste de 18 secteurs... Pour y voir plus clair en ce qui concerne têtes, cylindres et secteurs voir plus haut.

Cela crée une image de la disquette sur le disque dur, laissant intact l'info de boot.

Le second exemple utilise le "bs=" par défaut de 512, qui est la taille du secteur d'une disquette.

Copier l'image d'une disquette du disque dur vers une disquette

Faire :

```
dd if=/home/sam/floppy.image of=fd0 bs=2x80x18b conv=notrunc
```

Créer une disquette de démarrage

Vous pouvez créer une disquette de démarrage avec le fichier **boot.img**, qui est assez facile à obtenir. Vous avez seulement besoin d'un programme qui commencera l'écriture au secteur 1.

```
dd if=boot.img of=/dev/fd0 bs=1440k
```

Ceci crée une disquette bootable et vous pouvez ajouter des données dessus.

Formater une série de disquettes

Prendre une disquette formatée vide qui n'a jamais été utilisée et faites :

```
dd if=/dev/fd0 of=/home/sam/floppy.bin
```

ce qui crée une image de nouvelle disquette formatée sur votre disque dur, puis chargez dans le lecteur de disquettes l'une des disquettes que vous voulez formater et faites :

```
dd if=/home/sam/floppy.bin of=/dev/fd0
```

Cette disquette se retrouvera alors exactement dans l'état de la disquette inutilisée avec laquelle vous avez commencé.

Ne copier que le MBR et le secteur de boot d'une disquette

Vous le copierez dans un fichier image sur le disque dur ainsi :

```
dd if=/dev/fd0 of=/home/sam/MBRboot.image bs=512 count=2
Cela copie les 2 premiers secteurs de la disquette.
```

Réparer une disquette infectée par un cheval de Troie DRM

Insérer la disquette et faites :

```
dd if=/dev/null of=/dev/fd0 conv=notrunc
dd if=/home/sam/floppy.image of=/dev/fd0 conv=notrunc,noerror
```

Normalement, écrire des octets nuls sur les deux premiers secteurs d'une disquette rend la disquette complètement inutilisable. Elle ne peut même plus être formatée après un tel traitement ! Mais grâce à l'image d'une disquette neuve, inutilisée, que vous avez pris soin de faire plus haut (disons : *floppy.image*), vous pouvez réécrire correctement les deux premiers secteurs.

Créer une image de la partition d'une autre machine en réseau

Faire démarrer les deux machines avec le CD helix, juste pour être absolument sûr.

Sur la machine source :

```
dd if=/dev/hda bs=16065b | netcat targethost-IP 1234
```

Sur la machine cible :

```
netcat -l -p 1234 | dd of=/dev/hdc bs=16065b
```

Netcat est un programme, disponible par défaut sur presque toutes les installations Linux. C'est comme un couteau de l'armée suisse pour les réseaux. Dans l'exemple précédent, netcat et dd sont tubés l'un à la sortie de l'autre. Une des fonctions du noyau Linux est de faire des tubes. Le caractère pour le 'tube' ressemble à deux petits tirets l'un au-dessus de l'autre et verticaux ou à une barre verticale. Voici comment cette commande fonctionne : la taille du bloc correspond à un cylindre. bs=1065b équivaut à un cylindre sur un disque LBA. La commande dd est tubée vers netcat, qui prend pour argument l'adresse IP de la cible (comme 192.168.0.1 ou toute adresse avec un port ouvert) et le port que vous désirez utiliser (1234).

Alerte! n'appuyez pas maintenant sur la touche entrée. Appuyez sur la touche entrée de la machine cible, appuyez sur la touche entrée de la machine source

C'est un peu comme cela que Norton Ghost fonctionne pour copier l'image d'une partition sur une autre machine.

Faire une recherche de chaines de caractères dans une partition tout entière

Supposons que vous souhaitiez trouver si votre petite amie vous trompe, a des cybers amants, ou se conduit mal avec son ordinateur. Même si l'ordinateur est protégé par un mot de passe, vous pouvez booter avec le CD: http://www.efense.com/helix ou un autre live-CD et chercher la chaine de caractères dans la partition entière, en utilisant la commande grep :

```
dd if=/dev/sda2 bs=16065 | hexdump -C | grep 'je ne l'aime vraiment plus'
```

Cherchera dans toute la partition la chaîne de caractères spécifiée entre les guillemets (utilisez éventuellement les options de la commande grep : avec, disons, `grep -3 'chaine'` vous auriez un peu plus de contexte affiché pour chaque résultat de recherche). Chercher plusieurs fois dans une partition entière peut être fastidieux. Cette commande particulière sur les chaines de caractères imprime le résultat de la recherche sur l'écran, avec l'offset où il se trouve dans la partition. dd travaille en système décimal. L'offset des disques travaille en hexadécimal. Disons que vous trouvez la chaine de caractères dans votre partition à l'offset 0x020d0d90h. Vous convertissez cela en

décimal avec l'une des nombreuses calculatrices trouvées sur Linux. Ceci est l'offset décimal 34409872. Divisé par 512 par secteur, nous obtenons 67206,78125.

Rechercher dans des fichiers effacés

```
dd if=/dev/sda2 bs=16065 skip=2140 count=3 | less
```

Cette ligne de commande écrira à l'écran en évitant ainsi d'écrire accidentellement un fichier dans un espace libre du disque, qui peut contenir des fichiers effacés que vous désirez rechercher. Avec cette méthode, vous pouvez recherchez dans tous les fichiers effacés, toute activité de chat ou e-mails. Elle fonctionne quelle que soit la sécurité utilisée sur la machine. Elle fonctionne avec les partitions NTFS, ext2, ext3, reiserfs, swap, et FAT. **Mais il est illégal d'utiliser cette méthode sur un ordinateur pour lequel vous n'y êtes pas autorisé. On peut être poursuivi ou emprisonné pour avoir réalisé des recherches non autorisées.**

Notez tant qu'on y est que, vous pouvez copier la mémoire du système sur un CD. Ceci est utile pour décrire le contenu de la mémoire sans contaminer le disque dur. Je recommande d'utiliser un CD-RW pour vous exercer un peu. Ceci ne concerne pas dd, mais c'est cool.

```
cdrecord dev=ATAPI:0,1,0 -raw tsize=700000000 driveropts=burnfree /dev/mem
```

pour trouver le graveur CD

```
cdrecord -scanbus=ATAPI
```

dd ne copiera pas ou n'effacera pas une HPA (host protected area, zone hôte protégée de l'hôte). si correctement utilisée, dd effacera complétement un disque, mais pas aussi bien qu'en utilisant l'effacement matériel garanti, la commande d'effacement sécurisé.

Pour lire la mémoire ainsi enregistrée faites :

```
dd if=/dev/hdd | less
```

On fera une recherche en s'aidant de la commande grep :

```
dd if=/dev/hdd | hexdump -C | grep 'chaine_de_caracteres'
```

La chaine de caractères entre guillemets est n'importe quelle séquence ascci ou hexadécimale (doit être séparée avec un espace: '55 <espace>aa<espace>09' recherche la chaine hexadécimale '55aa09').

On peut utiliser pour cette recherche grep les classes de caractères POSIX :

[[:alnum:]] tout caractère alphanumérique

[[:alpha:]] tout caractère alphabétique

[[:digit:]] tout caractère numérique

[[:blank:]] tabulations et espaces

[[:lower:]] tout caractère alphabétique en minuscule

[[:upper:]] tout caractère alphabétique en majuscule

[[:cntrl:]] tout caractère ASCII de 000 à 037, et 177 octal

[[:graph:]] tout caractère [[:alnum:]] et [[:punct:]]

[[:punct:]] tout caractère de ponctuation ` ! ' # $ % ' ( ) * + - . / : ; < = > ? @ [\ ] ^ _ { | } ~

[[:space:]] tabulation, nouvelle ligne, tabulation verticale , nouvelle page, retour charriot, et espace

[[:xdigit:]] tout caractère admis dans un chiffre hexadécimal (0 à 9, a à f, A à F).

Sauvegarde de disquettes sur disque dur

Je sauvegarde toutes mes disquettes sur disque dur. Les disquettes ne sont pas éternelles, aussi je fais :

```
dd if=/dev/fd0 of=/home/sam/disquettes/backup.bin conv=notrunc
```

Si ma disquette défaille, je peux faire des copies en nombre illimité :

```
dd if=/home/sam/disquettes/backup.bin of=/dev/fd0 conv=notrunc
```

Lire le BIOS

```
dd if=/dev/mem bs=1k skip=768 count=256 2>/dev/null | strings -n 8
```

Pour aller plus loin

Récupérer des secteurs défectueux

Il existe une variante de dd pour récupérer des données sur un média défectueux, tel qu'un disque dur avec des mauvais secteurs. Il est appelé *dd_rescue*. Il est disponible ici : http://www.garloff.de/kurt/linux/ddrescue/

Nota Bene : Il existe aussi un paquetage rpm de *dd_rescue* pour Mandriva, installable depuis les sources usuelles.

L'implémentation de dd pour le département de la défense est appelée *dcfldd*, et possède quelques caractéristiques comme une barre de progression, ainsi vous pouvez planifier vos pauses café : http://dcfldd.sourceforge.net/

Sdd

Sdd est utile quand la taille des blocs d'entrée est différente de celle des blocs de sortie, et réussira dans des cas où dd échoue

http://linux.maruhn.com/sec/sdd.html

Un des meilleurs liens sur dd

http://www.softpanorama.org/Tools/dd.shtml

Pour plus de détails techniques

Les opérandes

Les opérandes suivants sont supportés :

if=fichier

Spécifie le chemin d'où proviennent les données entrantes. L'entrée standard est l'entrée par défaut.

of=fichier

Spécifie le chemin où sont dirigées les données en sortie. La sortie standard est la sortie par défaut.

ibs=n

Spécifie que la taille des blocs d'entrée est n octets (512 par défaut).

obs=n

Spécifie que la taille des blocs de sortie est n octets (512 par défaut).

bs=n

Spécifie la taille des blocs d'entrée et de sortie à n octets, en prenant le pas sur *ibs=* et *obs=*
Si aucune conversion autre que : *sync*, *noerror* et *notrunc* n'est spécifiée, chaque bloc d'entrée est copié dans la sortie en un seul bloc sans regroupement des petits blocs.

cbs=n

Spécifie la taille des blocs pour la conversion pour les opérandes *block* et *unblock*, en nombre d'octets (0 par défaut). Si *cbs* est omis ou donné égal à 0, alors utiliser *block* ou *unblock* produit des résultats non spécifiés. Cette option n'est utilisée que si la conversion ASCII ou EBCDIC est spécifiée.
Pour les opérandes *ascii* et *asciib*, les données d'entrée sont traitées comme décrit pour l'opérande *unblock* si ce n'est que les caractères sont convertis en ASCII avant que les caractères SPACE de remplissage ne soient effacés. Pour les opérandes *ebcdic*, *ebcdicb*, *ibm*, et *ibmb*, les données d'entrée sont traitées comme décrit pour l'opérande *block* si ce n'est que les caractères sont convertis en EBCDIC ou en IBM EBCDIC après l'ajout des caractères SPACE de remplissage.

files=n

Copie et concatène n fichiers d'entrée avant de terminer (n'a de sens que si l'entrée est une bande magnétique ou similaire).

skip=n

Saute n blocs d'entrée (en utilisant la taille de bloc d'entrée spécifiée par *bs* ou *ibs* ou la taille par défaut) avant de commencer la copie.
Sur les fichiers pointables (*seekable files*), l'implémentation lit les blocs ou pointe derrière eux. Sur les fichiers non pointables, les blocs sont lus et les données laissées de côté.

iseek=n

Se positionne n blocs après le début du fichier d'entrée avant de copier (pertinent pour les fichiers disques, où skip peut être terriblement lent).

oseek=n

Se positionne n blocs après le début du fichier de sortie avant de copier

seek=n

Saute n blocs (en utilisant la taille de bloc de sortie spécifiée par *bs* ou *obs* ou la taille par défaut) après le début du fichier de sortie avant de commencer la copie.

Sur les fichiers non pointables (non *seekable files*), les blocs existants sont lus et l'espace compris entre l'actuel fin de fichier et l'offset spécifié, s'il existe, est rempli avec des octets nuls. Sur les fichiers pointables (*seekable files*), l'implémentation pointe sur l'offset spécifié ou lit les blocs comme décrit pour les fichiers non pointables.

count=n

Ne copie que n blocs d'entrée.

conv=valeur[,valeur. . .]
Où les *valeurs* sont des symboles séparés par des virgules, tirés de la liste suivante :

ascii

Convertit EBCDIC en ASCII.

asciib

Convertit EBCDIC en ASCII en utilisant les translations de caractères compatibles avec BSD.

ebcdic

Convertit ASCII en EBCDIC. Lors d'une conversion d'enregistrements ASCII de longueur fixe sans caractères NEWLINE (nouvelle ligne), commence par créer un tube avec : `dd conv=unblock`.

ebcdicb

Convertit ASCII en EBCDIC en utilisant les translations de caractères compatibles avec BSD. Lors d'une conversion d'enregistrements ASCII de longueur fixe sans caractères NEWLINE (nouvelle ligne), commence par créer un tube avec : `dd conv=unblock`.

ibm

Conversion légèrement différente de ASCII en EBCDIC. Lors d'une conversion d'enregistrements ASCII de longueur fixe sans caractères NEWLINE (nouvelle ligne), commence par créer un tube avec : `dd conv=unblock`.

ibmb

Conversion légèrement différente de ASCII en EBCDIC en utilisant les translations de caractères compatibles avec BSD. Lors d'une conversion d'enregistrements ASCII de longueur fixe sans caractères NEWLINE (nouvelle ligne), commence par créer un tube avec : `dd conv=unblock`.

Les valeurs *ascii* (ou *asciib*), *ebcdic* (ou *ebcdicb*), et *ibm* (ou *ibmb*) sont mutuellement exclusives.

block

Les données d'entrée sont considérées comme une suite d'enregistrements de longueur variable terminés par un caractère NEWLINE (fin de ligne) ou par un caractère EOF (fin de fichier), indépendamment des frontières entre blocs dans les données d'entrée. Chaque enregistrement est ensuite converti en un enregistrement ayant une longueur fixe, spécifiée par la taille de bloc pour la conversion (voir *cbs=*). Tous les caractères NEWLINE sont effacés de la ligne d'entrée. Des caractères SPACE (espace) sont ajoutés aux lignes dont la taille est inférieure à celle de leur taille de bloc pour la conversion. Les lignes plus longues que la taille de bloc pour la conversion sont tronquées de façon à ce qu'elles contiennent le plus grand nombre possible de caractères pouvant tenir dans cette taille. Le nombre des lignes tronquées est signalé.

unblock

Convertit les enregistrements de taille fixe en enregistrements de taille variable. Lit un nombre d'octets égal à la taille de bloc pour la conversion spécifié par *cbs=* (ou le nombre d'octets restant en

entrée, s'il est inférieur à la taille de bloc pour la conversion), efface tous les caractères SPACE (espace) de remplissage en fin d'enregistrement et ajoute un caractère NEWLINE (fin de ligne).

Les valeurs *block* et *unblock* sont mutuellement exclusives.

lcase

Transforme les majuscules en minuscules lorsque la locale définie par LC_CTYPE le permet. Les caractères pour lesquels aucune correspondance majuscules/minuscules n'est spécifiée dans la locale ne sont pas modifiés par cette conversion.

ucase

Transforme les minuscules en majuscules lorsque la locale définie par LC_CTYPE le permet. Les caractères pour lesquels aucune correspondance majuscules/minuscules n'est spécifiée dans la locale ne sont pas modifiés par cette conversion.

swab

Permute chaque paire d'octets d'entrée. Si l'enregistrement d'entrée en cours possède un nombre impair d'octets, le dernier octet de l'enregistrement d'entrée est ignoré.

noerror

N'arrête pas le processus en cas d'erreur d'entrée. Lorsque survient une erreur d'entrée, un message de diagnostic est envoyé vers la sortie d'erreur standard, suivi par le décompte en cours des bloc d'entrée et de sortie dans le même format que celui utilisé en complétion. Si la conversion *sync* est spécifiée, les données manquantes sont remplacées par des octets nuls qui sont traités normalement. Autrement, le bloc d'entrée sera omis de la sortie. *notrunc* ne tronque pas le fichier de sortie. Dans le fichier de sortie les blocs non explicitement écrits par cette invocation de dd seront préservés. (Voir aussi le précédent opérande : of=file).

notrunc

Le fichier de sortie ne subit pas de troncation (en son absence les blocs de données formés d'octets à zéro sont remplacés par une chaîne de cinq astérisques).

sync

Assemble chaque bloc d'entrée à la taille du tampon *ibs=*, ajoutant des octets nuls. (Si soit *block* soit *unblock* est aussi specifié, ajoute des caractères SPACE, à la place d'octets nuls.)

Si des opérandes autres que *conv=* sont spécifiés plus d'une fois, le dernier *operande=valeur* spécifié est utilisé.

Pour les opérandes *bs=*, *cbs=*, *ibs=*, et *obs=*, l'application doit fournir une expression qui spécifie une taille en octets. L'expression, *expr*, peut être :

un nombre décimal positif
un nombre décimal positif suivi de **k**, spécifiant ainsi une multiplication par 1024
un nombre décimal positif suivi de **M**, spécifiant ainsi une multiplication par 1024*1024
un nombre décimal positif suivi de **b**, spécifiant ainsi une multiplication par 512
deux ou plusieurs nombres décimaux positifs (avec ou sans k ou b) séparés par **x**, spécifiant ainsi le produit des valeurs indiquées.

Les variables d'environnement

Les variables d'environnement suivantes affectent les messages et les messages d'erreur de dd :

LANG

Fournit une valeur par défaut pour les variables d'internationalisation qui sont non définies ou nulles. Si LANG est non défini ou nul, la valeur correspondante par défaut dépendante de l'implantation locale sera utilisée. Si chacune des variables d'internalisation contient une valeur non valide, l'utilitaire se comportera comme si aucune des variables n'avait été définie.

LC_ALL

Si positionnée sur une valeur de chaine de caractères non vide, prend le pas sur les valeurs de toutes les autres variables d'internationalisation.

LC_CTYPE

Définit la locale utilisée pour l'interprétation comme caractères des séquences d'octets de données texte (par exemple, caractères à octet unique, par opposition aux caractères multi-octets dans les arguments ou les fichiers d'entrée), la classification des caractères en majuscules ou minuscules, et le mappage des caractères d'une casse à une autre.

LC_MESSAGES

Détermine la locale à utiliser pour déterminer le format et le contenu des messages de diagnostic écrits dans la sortie standard des erreurs et les messages informatifs écrits dans la sortie standard.

NLSPATH

Détermine l'endroit des catalogues de messages pour le traitement de LC_MESSAGES.

Obtenir un nombre d'enregistrements lus

Remarquer que l'envoi d'un signal SIGUSR1 à une commande dd en cours provoque l'écriture dans la sortie standard des erreurs du nombre d'enregistrements lus et écrits jusqu'à présent, puis la reprise de la copie.

```
$ dd if=/dev/zero of=/dev/null& pid=$!

$ kill -USR1 $pid; sleep 1; kill $pid

10899206+0 records in 10899206+0 records out
```